Carnegie Mellon

Mining Billion-Scale Graphs: Patterns and Algorithms

Christos Faloutsos and U Kang

CMU

Part 2: Algorithms

Complementary to tutorial: *Mining Billion-Scale Graphs: Systems and Implementations*: Haixun Wang et al

Part 2: Algorithms

Carnegie Mellon

Outline

- Problem#1: Patterns in graphs
- Problem#2: Tools
- Problem#3: Scalability - PEGASUS
  - Structure Analysis
  - Eigensolver
  - Graph Layout and Compression
- Conclusions

Our goal:

Open source system for mining huge graphs:

PEGASUS project (PEta GrAph mining System)

- www.cs.cmu.edu/~pegasus
- code and papers

Scalability Challenge

- The sizes of graphs are growing!

facebook

0.5 billion users
60 TBytes/day
15 PBytes/total
[Thusoo+ '10]

bing

ClickStream Data
0.26 PBytes
1 billion query-URL
[Liu+ '09]

YAHOO!

1.4 billion web pages
6.6 billion edges
[Broder+ '04]

Google

20 PBytes/day

[Dean+ '08]

Scalability Challenge

- The sizes of graphs are growing!

Q: How can we handle large graphs which don't fit into the memory, or disks of a single machine?

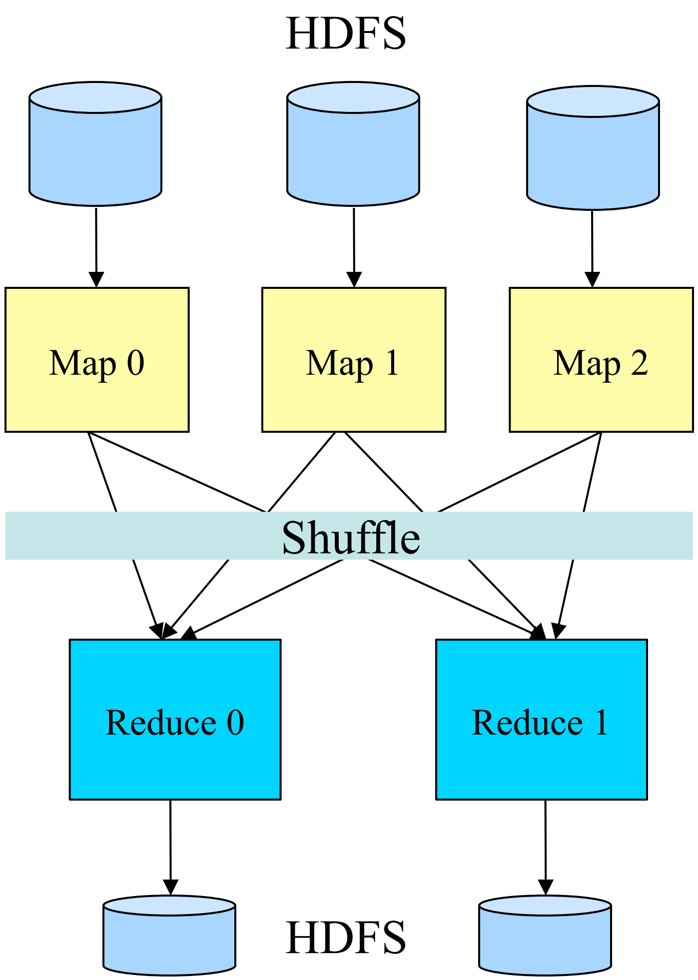
A: Parallelism, with MapReduce!

[Dean+ '08]

Carnegie Mellon

Background: MapReduce

- MapReduce/Hadoop Framework



HDFS: fault tolerant, scalable, distributed storage system

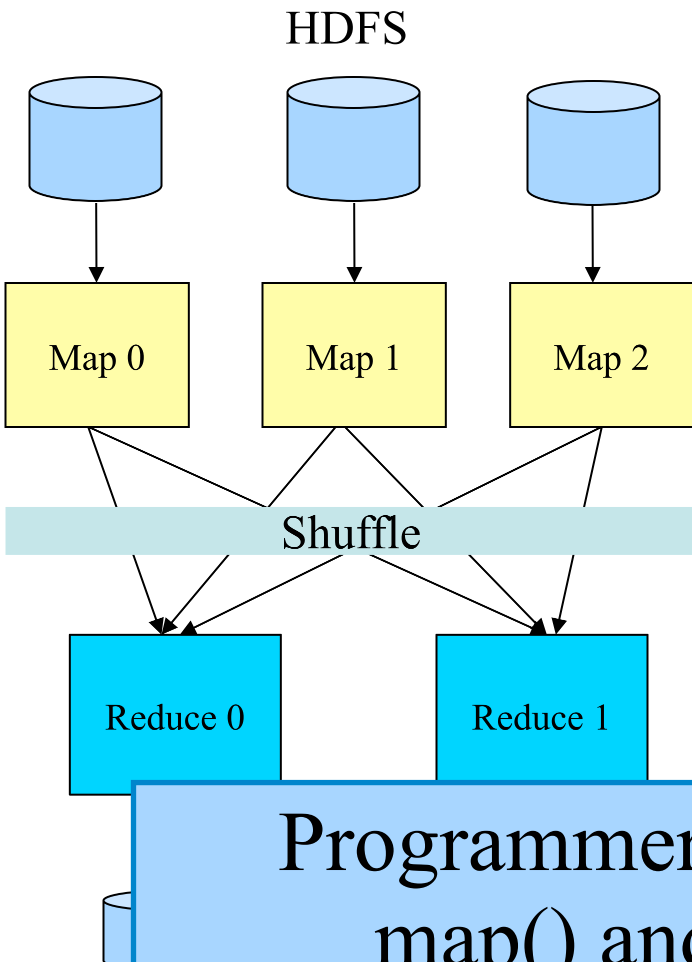
Mapper: read data from HDFS, output (k,v) pair

Output sorted by the key

Reducer: read output from mappers, output a new (k,v) pair to HDFS

Background: MapReduce

- MapReduce/Hadoop Framework



HDFS: fault tolerant, scalable, distributed storage system

Mapper: read data from HDFS, output (k,v) pair

Output sorted by the key

Reducer: read output from mapp ers, output a new (k,v) pair to H

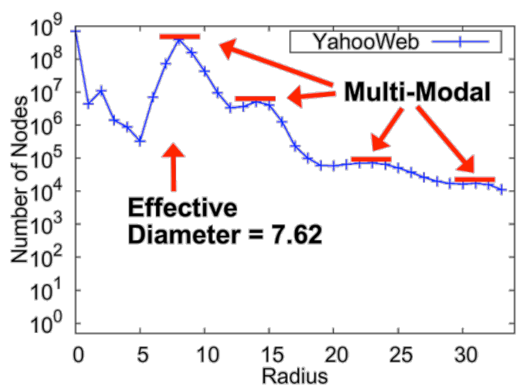
Programmers need to provide only map() and reduce() functions

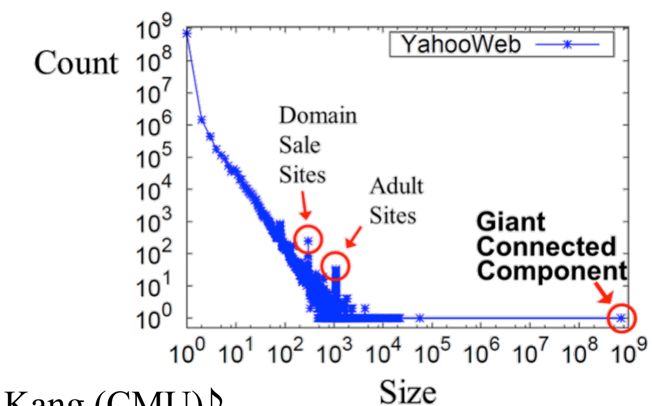
Outline

- Problem#1: Patterns in graphs
- Problem#2: Tools
- Problem#3: Scalability - PEGASUS
  - Structure Analysis
  - Eigensolver
  - Graph Layout and Compression
- Conclusions

Structure Analysis

- How to scale-up structure analysis algorithm?
  - Q1: How to **unify** many structure analysis algorith ms (connected components, PageRank, diameter/ra dius)?
  - Q2: How to design a **scalable** algorithm for the stru cture analysis?





Q1: Unifying Algorithms

- Given a graph, can we compute
  - connected components,
  - PageRank,
  - Random Walk with Restart,
  - diameter/radius

  with ***one algorithm***?

Q1: Unifying Algorithms

- Given a graph, can we compute
  - connected components,
  - PageRank,
  - Random Walk with Restart,
  - diameter/radius

  with ***one algorithm***?

Yes!

How ?

Carnegie Mellon

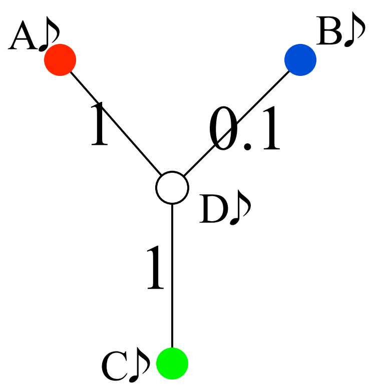
Main Idea

- GIM-V
  - Generalized Iterative Matrix-Vector Multiplication
  - Extension of plain matrix-vector multiplication
  - includes
    - Connected Components
    - PageRank
    - RWR (Random Walk With Restart)
    - Diameter Estimation

Carnegie Mellon

Main Idea: Intuition

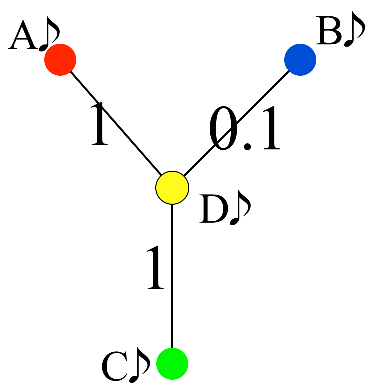
- Plain M-V multiplication



- Weighted Combination of Colors
- ~ Message Passing

Main Idea: Intuition

- Plain M-V multiplication

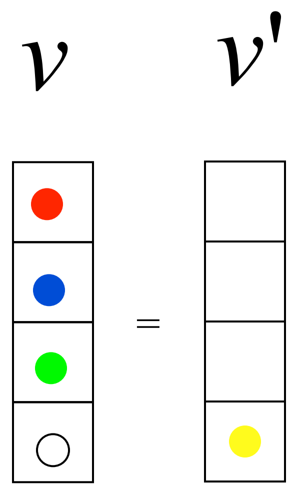


- Weighted Combination of Colors
- ~ Message Passing

M

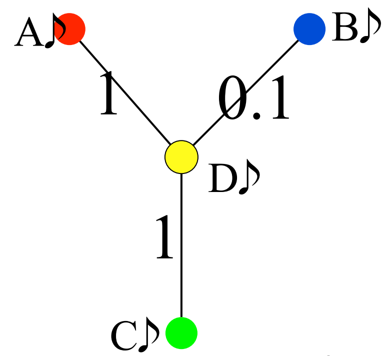
	A	B	C	D
A				1
B				1
C				0.1
D	1	1	0.1	

X



$$v_4' = \sum_{i=1}^{4} m_{4i} v_i$$

Main Idea: Intuition

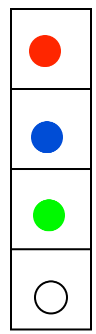


- Plain M-V multiplication

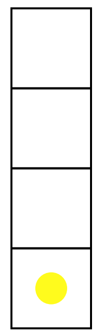
$$M \qquad\qquad v \qquad\qquad v'$$

	A	B	C	D
A				1
B				1
C				0.1
D	1	1	0.1	

X



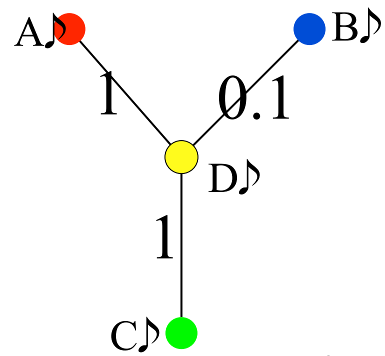
=



$$M \times v = v'$$

$$v_j' = \sum_{i=1}^{4} m_{ji} v_i$$

Main Idea: Intuition



- Plain M-V multiplication

M v v'

	A	B	C	D
A				1
B				1
C				0.1
D	1	1	0.1	

X =

$$M \times v = v'$$

$$v_j' = \sum_{i=1}^{4} m_{j\,i} v_i$$

Three Implicit Operations here:

multiply $m_{j\,i}$ and v_i	combine2	Message sending
sum n multiplication results	combineAll	Message combination
update v_j'	assign	

Main Idea

- GIM-V
  - Generalizing the three operations leads to many algo rithms

operations	Standard MV	Con. Cmpt.	PageRank	RWR	Diameter
combine2	Multiply				
combineAll	Sum		?		
assign	Assign				

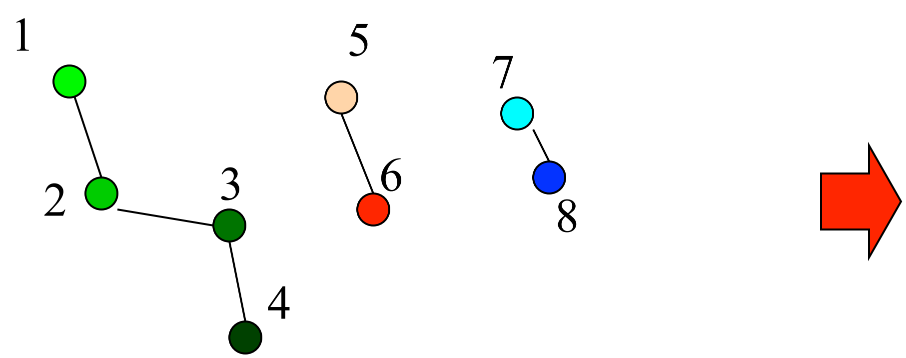
Main Idea

- GIM-V for Connected Components

operations	Standard MV	Con. Cmpt.	PageRank	RWR	Diameter
combine2	Multiply	Bool. X			
combineAll	Sum	MIN			
assign	Assign	MIN			

Main Idea

- GIM-V for Connected Components
  - How many connected components?
  - Which node belong to which component?



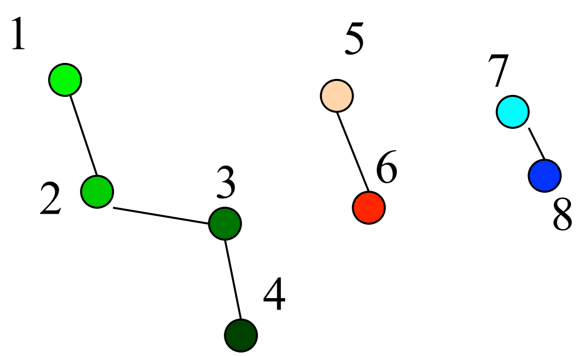
Input Graph

node id	component id
1	1
2	1
3	1
4	1
5	5
6	5
7	7
8	7

Output

Main Idea

- GIM-V for Connected Components

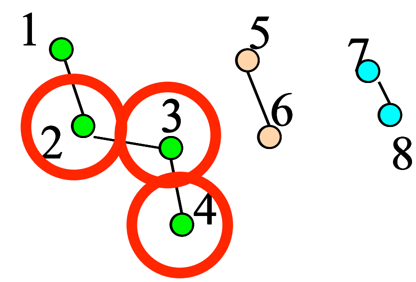


	1	2	3	4	5	6	7	8
1		1						
2	1		1					
3		1		1				
4			1					
5						1		
6					1			
7								1
8							1	

$$\times_G \begin{bmatrix}1\\2\\3\\4\\5\\6\\7\\8\end{bmatrix} \overset{?}{=} \begin{bmatrix}1\\1\\1\\1\\5\\5\\7\\7\end{bmatrix}$$

initial vector final vector

Main Idea



- GIM-V for Connected Components

$$combine2(m_{i,j}, v_j) = m_{i,j} \times v_j$$ "Sending Invitations"

$$combineAll(x_1, \ldots, x_n) = \min\{x_i \mid i = 1..n\}$$ "Accept the Smallest"

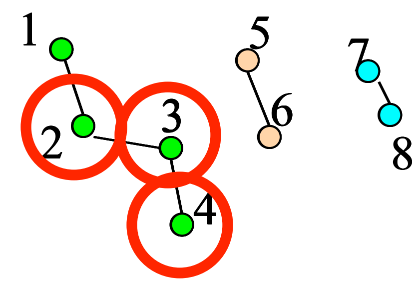
$$assign(v_i, v_{new}) = \min(v_i, v_{new})$$

	1	2	3	4	5	6	7	8
1		1						
2	1		1					
3		1		1				
4			1					
5						1		
6					1			
7								1
8							1	

$\times_G$

		=		=			
1	=	min(1, min(2))	=	1	1	1
2		min(2, min(1,3))		1	1	1
3		min(3, min(2,4))		2	1	1
4		min(4, min(3))		3	2	1
5		min(5, min(6))		5	5	5
6		min(6, min(5))		5	5	5
7		min(7, min(8))		7	7	7
8		min(8, min(7))		7	7	7

Main Idea



- GIM-V for Connected Components

$$combine2(m_{i,j}, v_j) = m_{i,j} \times v_j$$ “Sending Invitations”

$$combineAll(x_1, \ldots, x_n) = \min\{x_i \mid i = 1..n\}$$ “Accept the Smallest”

$$assign(v_i, v_{new}) = \min(v_i, v_{new})$$

	1	2	3	4	5	6	7	8
1		1						
2	1		1					
3		1		1				
4			1					
5						1		

$\times_G$

1
2
3
4
5

=

min(1, min(2))
min(2, min(1,3))
min(3, min(2,4))
min(4, min(3))
min(5, min(6))

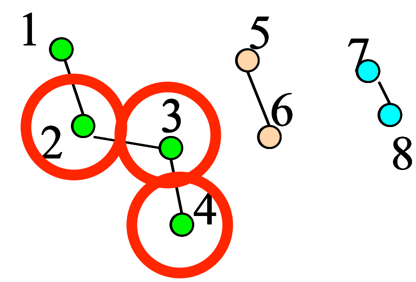
=

1	1	1
1	1	1
2	1	1
3	2	1
5	5	5
		5
		7
		7

1 GIM-V with MIN() operation
= find minimum node ids within 1 hop

Carnegie Mellon

Main Idea



- GIM-V for Connected Components

$$combine2(m_{i,j}, v_j) = m_{i,j} \times v_j$$ "Sending Invitations"

$$combineAll(x_1, \ldots, x_n) = \min\{x_i \mid i = 1..n\}$$ "Accept the Smallest"

$$assign(v_i, v_{new}) = \min(v_i, v_{new})$$

	1	2	3	4	5	6	7	8
1		1						
2	1		1					
3		1		1				
4			1					
5						1		

$\times_G$

1
2
3
4
5

=

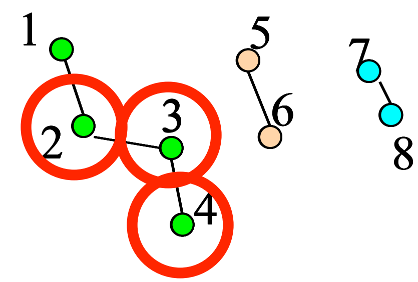
min(1, min(2))
min(2, min(1,3))
min(3, min(2,4))
min(4, min(3))
min(5, min(6))

=

1	1	1
1	1	1
2	1	1
3	2	1
5	5	5
		5
		7
		7

(k) GIM-V with MIN() operation
= find minimum node ids within (k) hops

Main Idea



- GIM-V for Connected Components

$$combine2(m_{i,j}, v_j) = m_{i,j} \times v_j$$ “Sending Invitations”

$$combineAll(x_1, \ldots, x_n) = \min\{x_i \mid i = 1..n\}$$ “Accept the Smallest”

$$assign(v_i, v_{new}) = \min(v_i, v_{new})$$

	1	2	3	4	5	6	7	8
1		1						
2	1		1					
3		1		1				
4			1					
5						1		
6					1			

$\times_G$

1	=	min(1, min(2))	=	1	1	1
2		min(2, min(1,3))		1	1	1
3		min(3, min(2,4))		2	1	1
4		min(4, min(3))		3	2	1
5		min(5, min(6))		5	5	5
6		min(6, min(5))		5	5	5
						7
						7

Maximum # of iterations : diameter

Carnegie Mellon

Main Idea

- GIM-V for PageRank

Operations	Standard MV	Con. Cmpt.	PageRank	RWR	Diameter
combine2	Multiply	Multiply	**Multiply with c**		
combineAll	Sum	MIN	**Sum with rj prob**		
assign	Assign	MIN	**Assign**		

Carnegie Mellon

Main Idea

Details

- GIM-V: PageRank
  - PageRank vector p: eigenvector of A:

$$1p = A \times p$$

nx1 nxn nx1

- Where

$$A = cE^{T} + (1 - c)U$$

Adjacency Matrix

All elements set to 1/n

Carnegie Mellon

Main Idea

Details

- GIM-V: PageRank
  - Algorithm: Power method

    (many multiplications)

$$p_{next} \leftarrow A \times p_{cur}$$

nx1 nxn nx1

Main Idea

Details

- GIM-V: PageRank

$$p_{next} \leftarrow E^T \times_G p_{cur}$$

nx1 nxn nx1

Algorithm: Power method
(many multiplications)

$$combine2(m_{i,j}, v_j) = c \times m_{i,j} \times v_j$$
$$combineAll(x_1, \ldots, x_n) = \tfrac{1-c}{n} + \sum x_i$$
$$assign(v_i, v_{new}) = v_{new}$$

Carnegie Mellon

Main Idea

- GIM-V

(approx.)

Operations	Standard MV	Con. Cmpt.	PageRank	RWR	Diameter
combine2	Multiply	Multiply	Multiply with c	Multiply with c	Multiply bit-vector
combineAll	Sum	MIN	Sum with rj prob.	Sum with restart prob	BIT-OR()
assign	Assign	MIN	Assign	Assign	BIT-OR()

Two Restrictions on HDFS

- [R1] HDFS is location transparent
  - Users don't know which file is located in which ma chine
- [R2] A line is never split
  - A large file is split into pieces of a size(e.g. 256 M B)
  - Users don't know the point of the split

Q2: Fast Algorithms for GIM-V

- Given the two restrictions R1 and R2, how can we make faster algorithms for GIM-V in Hadoop?
  - Three main ideas:
    - I1) Block Multiplication
    - I2) Clustering
    - I3) Compression

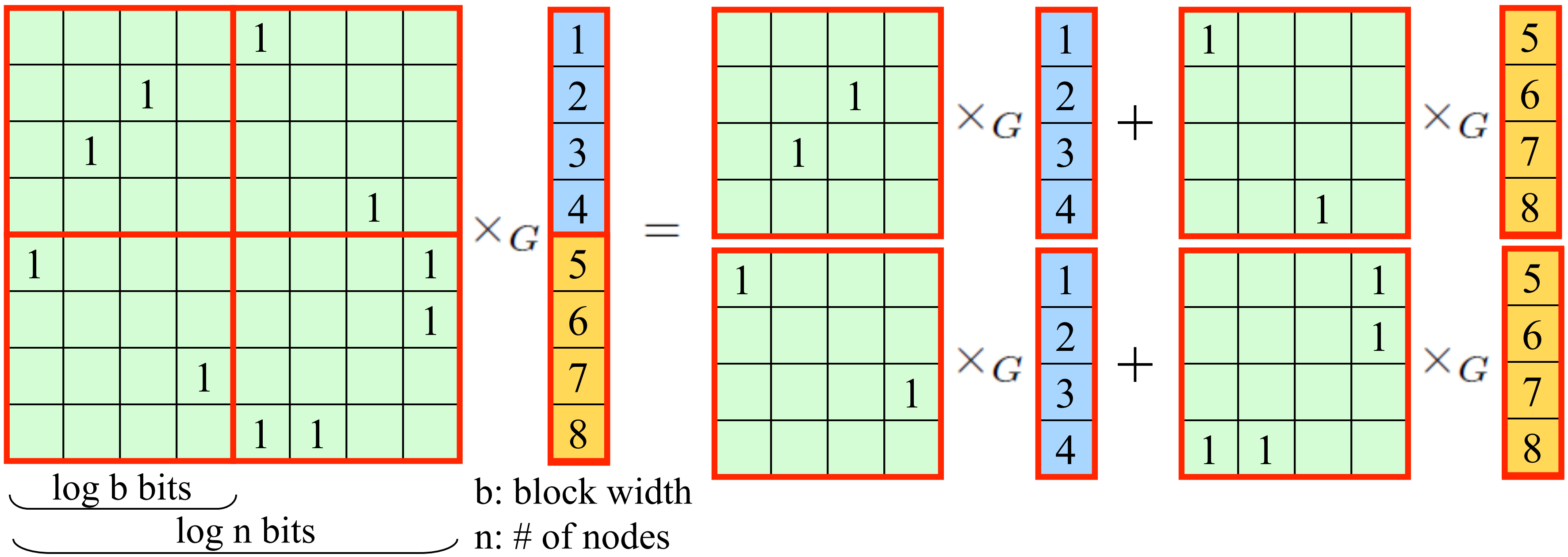
Main Idea

- I1) Block-Method

$$
\begin{bmatrix}
 & & & & 1 & & & \\
 & & 1 & & & & & \\
 & 1 & & & & & & \\
 & & & & & & 1 & \\
1 & & & & & & & 1 \\
 & & & & & & & 1 \\
 & & & 1 & & & & \\
 & & & & 1 & 1 & &
\end{bmatrix}
\times_G
\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}
=
\begin{matrix}
\begin{bmatrix}
 & & & \\
 & & 1 & \\
 & 1 & & \\
 & & &
\end{bmatrix}
\times_G
\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}
+
\begin{bmatrix}
1 & & & \\
 & & & \\
 & & & \\
 & & 1 &
\end{bmatrix}
\times_G
\begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}
\\
\begin{bmatrix}
1 & & & \\
 & & & \\
 & & & 1 \\
 & & &
\end{bmatrix}
\times_G
\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}
+
\begin{bmatrix}
 & & & 1 \\
 & & & 1 \\
 & & & \\
1 & 1 & &
\end{bmatrix}
\times_G
\begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}
\end{matrix}
$$

Main Idea

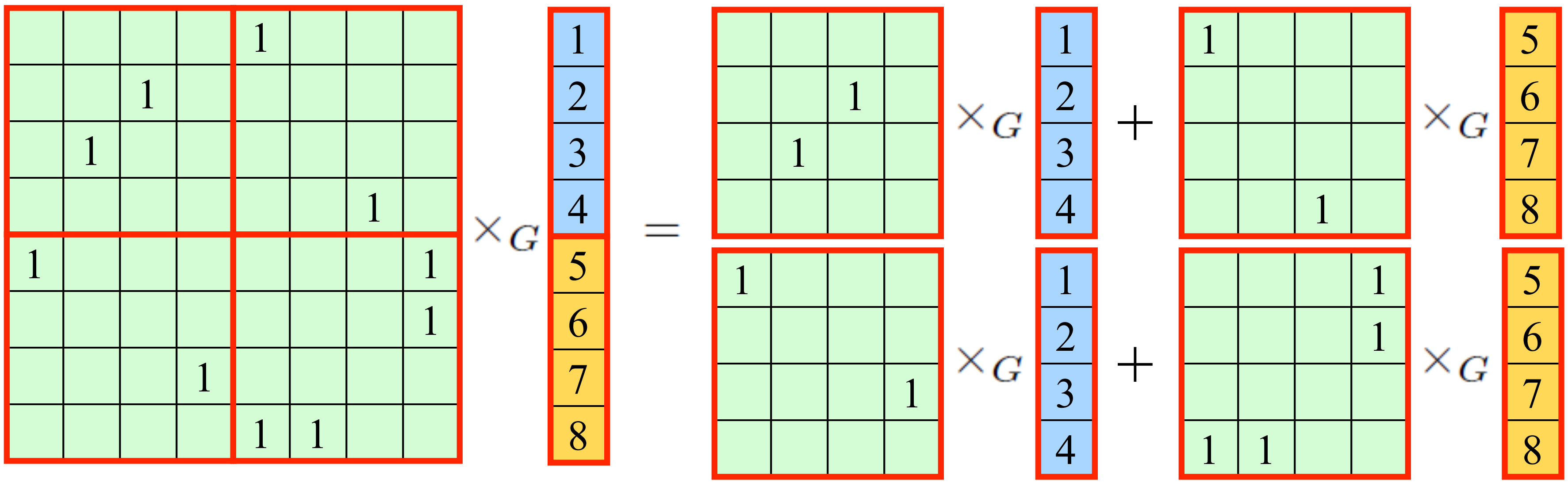
- I1) Block-Method



1. Group elements together into 1 line
2. Storage for an element: 2log n bits -> 2log b bits
3. Adjust the MapReduce code(block multiplication)

Main Idea

- I1) Block-Method



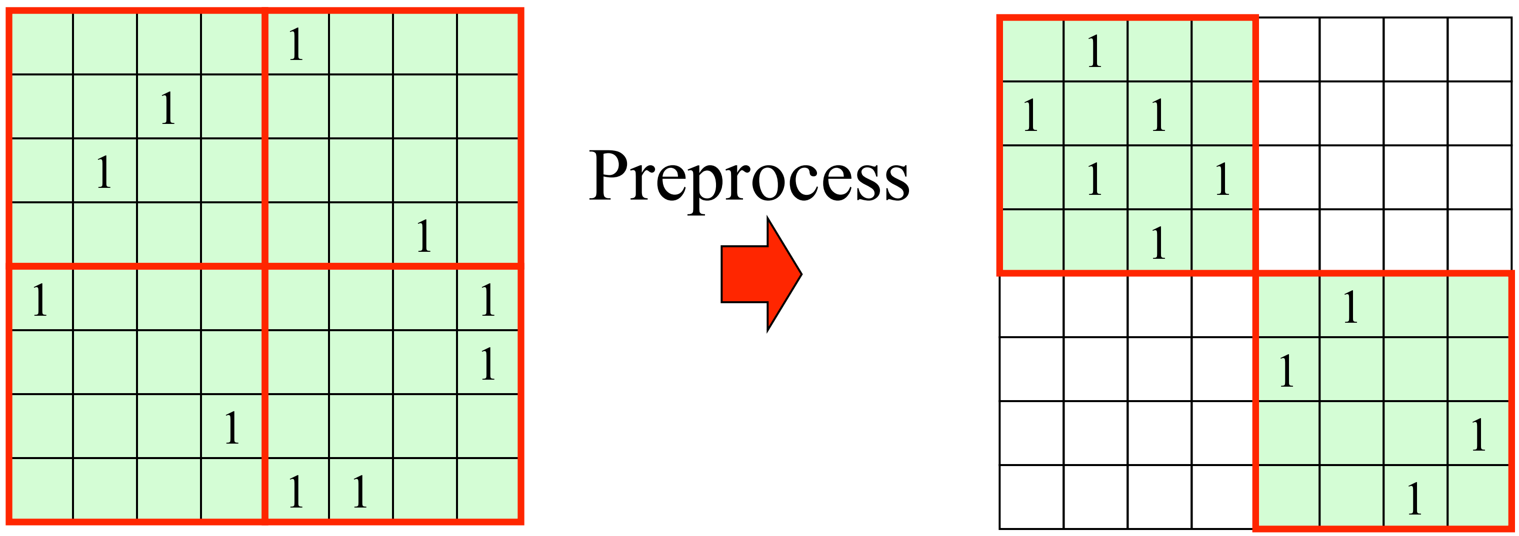
Thanks to the encoding,
- file size is decreased,
- shuffle time is decreased.

Main Idea

Q: Can we do even better?

Main Idea

- I2) Clustering

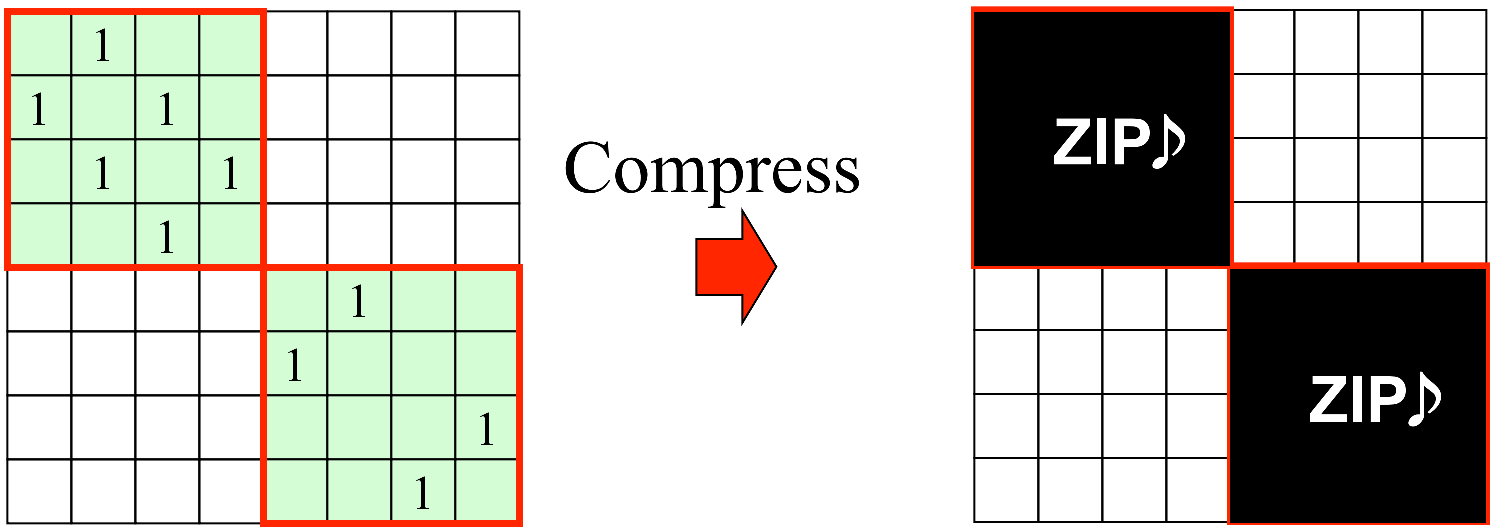


A: preprocessing for clustering
(only green blocks are stored in HDFS)

Carnegie Mellon

Main Idea

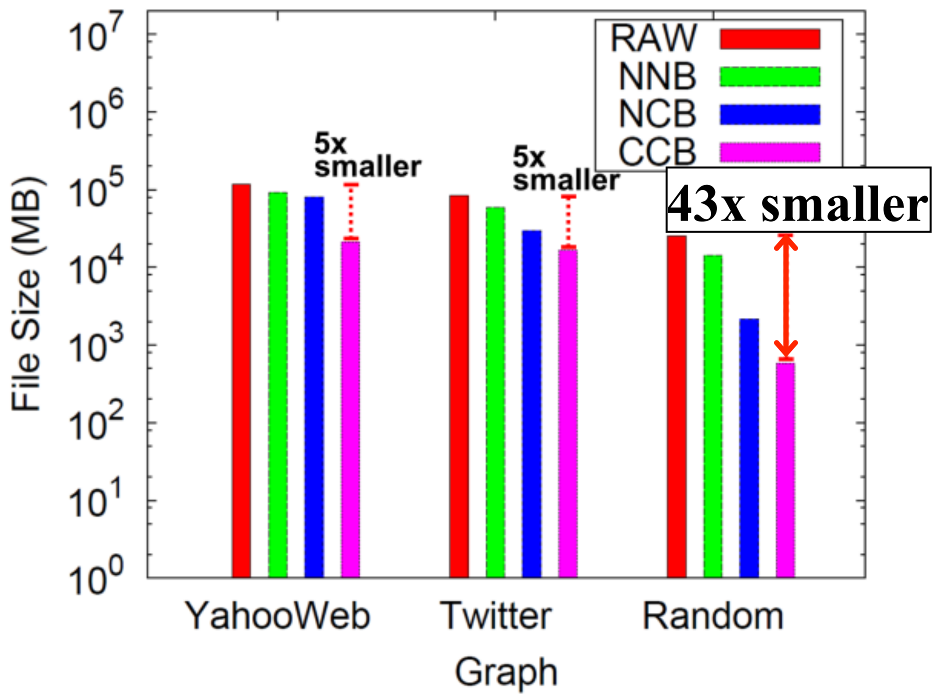
- I3) Compression

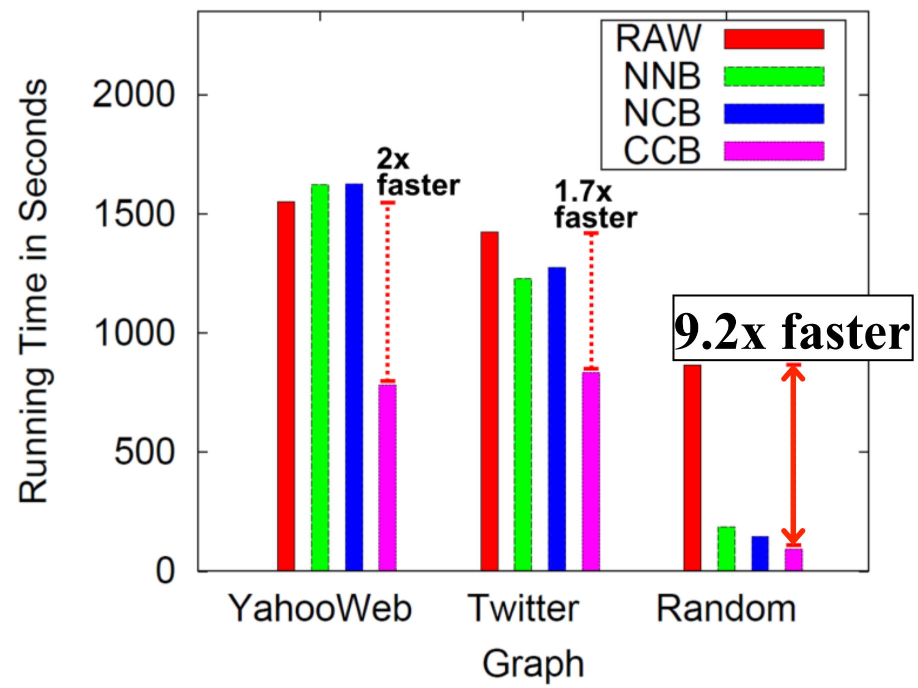


A: compress clustered blocks

Carnegie Mellon

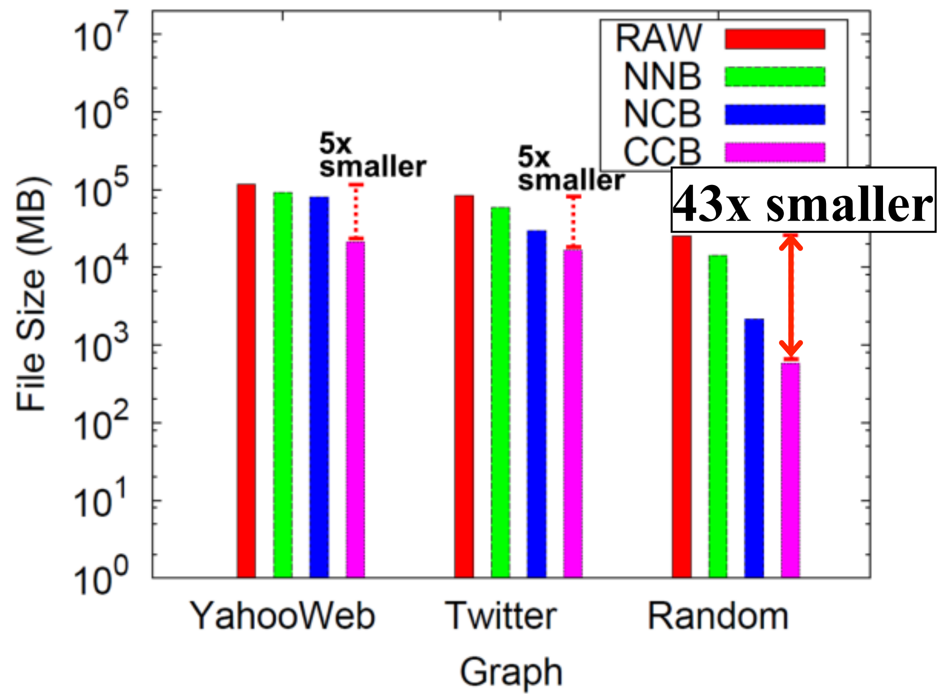
Result

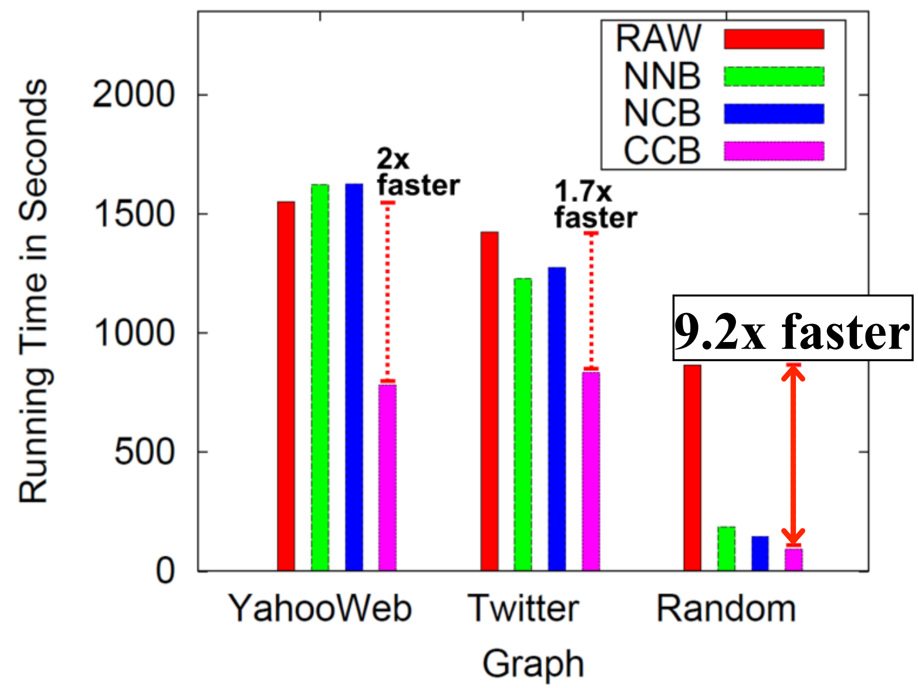




	Block Encoding ?	Compression ?	Clustering?
RAW	No	No	No
NNB	Yes	No	No
NCB	Yes	Yes	No
CCB	Yes	Yes	Yes

Result





A: Proposed Method(CCB) provides
43x smaller storage, 9.2x faster running time

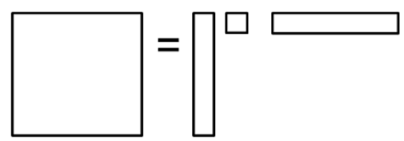
Carnegie Mellon

Outline

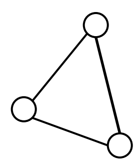
- Problem#1: Patterns in graphs
- Problem#2: Tools
- Problem#3: Scalability - PEGASUS
  - Structure Analysis
  - Eigensolver
  - Graph Layout and Compression
- Conclusions

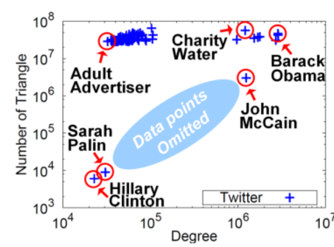
Background: Eigensolver

- Eigensolver
  - Given: (adjacency) matrix A,
  - Compute: top k eigenvalues and eigenvectors of A
  - Application:
    - SVD

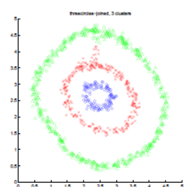


    - triangle counting





    - spectral clustering



Carnegie Mellon

Problem Definition

- Q4: How to design a billion-scale eigensolver?
  - Existing eigensolver: can handle millions of nodes a nd edges

Carnegie Mellon

Efficient Eigensolver

Details

- Lanczos Iterations

$\beta_0 = 0$, $q_0 = 0$, $b =$arbitrary, $q_1 = b/||b||$

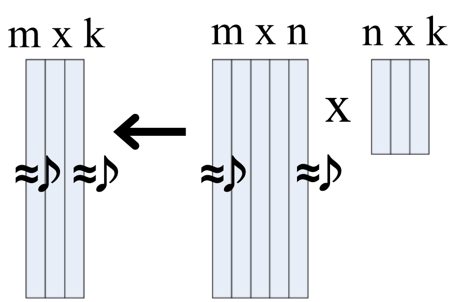
for $n = 1, 2, 3, \ldots$

$$v = Aq_n$$
$$\alpha_n = q_n^T v$$
$$v = v - \beta_{n-1}q_{n-1} - \alpha_n q_n$$
$$\beta_n = ||v||$$
$$q_{n+1} = v/\beta_n$$

1 matrix-vector multiplication
per iteration

Proposed Method

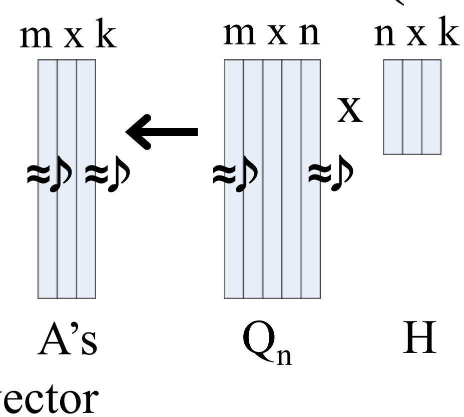
- HEigen algorithm (Hadoop Eigen-solver)
  - Selectively parallelize 'Lanczos-SO' algorithm
  - Block encoding
  - Exploiting skewness in matrix-matrix mult.
    - (m >> n > k)



Skewed Matrix-Matrix Mult.

Details

- Multiply Q_n^{mxn} and H^{nxk} (m >> n > k)



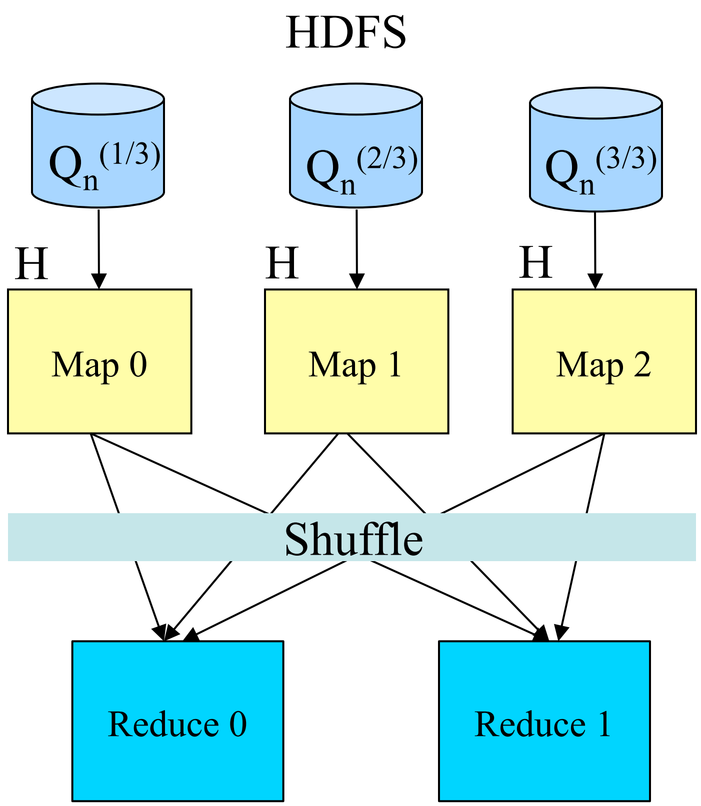
Q_n: O(100 Gbytes)
H: O(Kbytes)

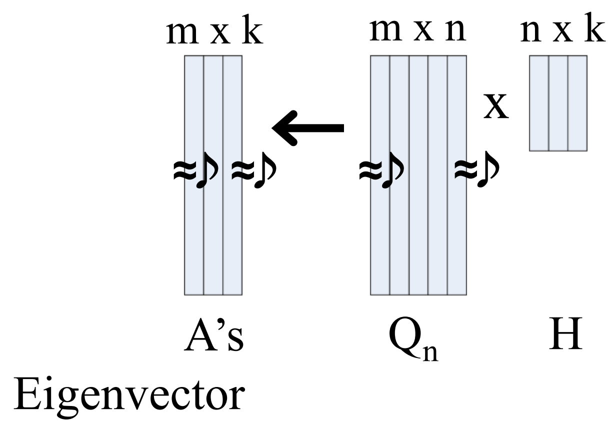
- Naïve multiplication: too expensive
- Proposed:
  - `cache'-based multiplication: broadcast the small matri x H to all the machines that contains Q_n

Skewed Matrix-Matrix Mult.

Details

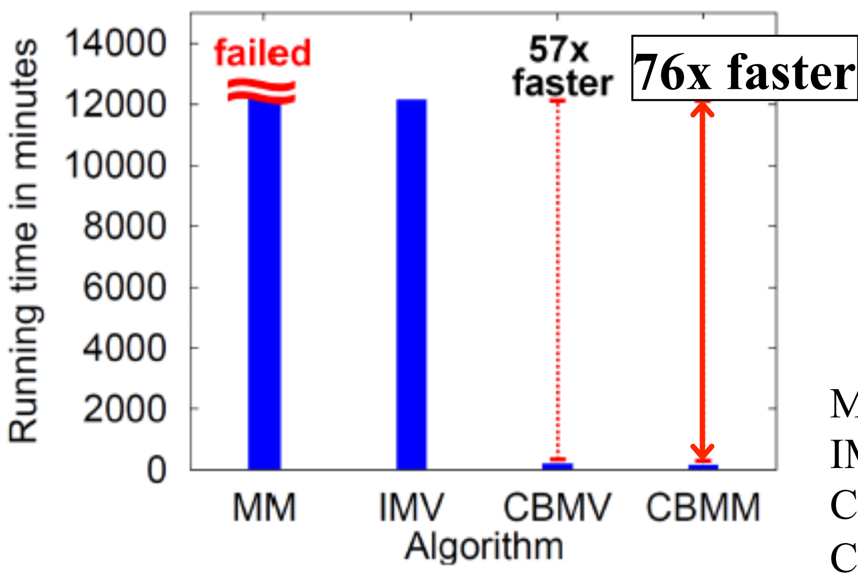
- `cache'-based multiplication: broadcast the small matrix H to all the machines that contains Q_n



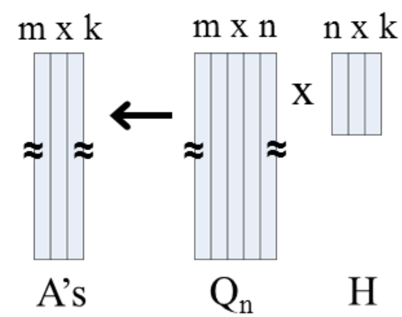


Skewed Matrix-Matrix Mult.

Which Matrix-Matrix multiplication algorithm runs the fastest?



Time vs. algorithms



Eigenvector

MM: naïve mat-mat mult.
IMV: naïve iterative mat-vec mult.
CBMV: cache-based iterative mat-vec mult.
CBMM: cache-based mat-mat mult.

(100 machines used)

Cache-based MM runs 76x faster

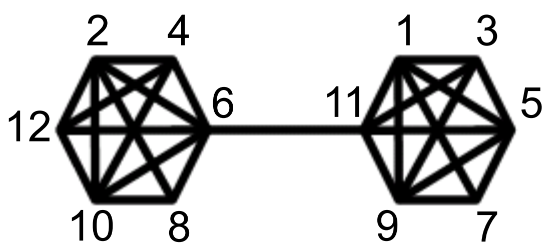
Carnegie Mellon

Outline

- Problem#1: Patterns in graphs
- Problem#2: Tools
- Problem#3: Scalability - PEGASUS
  - Structure Analysis
  - Eigensolver
  - Graph Layout and Compression
- Conclusions

Node Order Matters

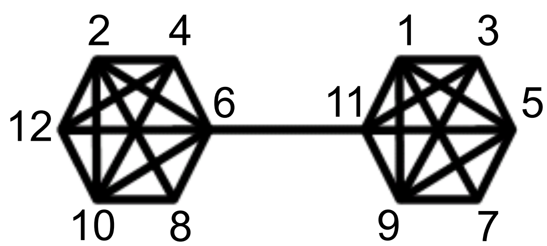
- A graph and the adjacency matrix



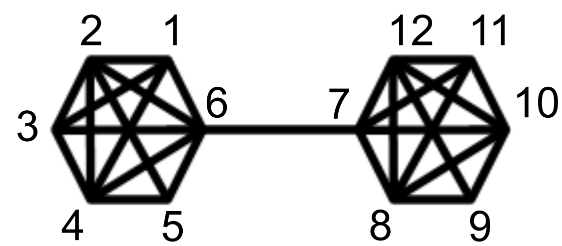
	1	2	3	4	5	6	7	8	9	10	11	12
1			1		1		1		1		1	
2				1		1		1		1		1
3	1				1		1		1		1	
4		1				1		1		1		1
5	1		1				1		1		1	
6		1		1				1		1	1	1
7	1		1		1				1		1	
8		1		1		1				1		1
9	1		1		1		1				1	
10		1		1		1		1				1
11	1		1		1	1	1		1			
12		1		1		1		1		1		

Node Order Matters

- Same graphs with different orderings



	1	2	3	4	5	6	7	8	9	10	11	12
1			1		1		1		1		1	
2				1		1		1		1		1
3	1				1		1		1		1	
4		1				1		1		1		1
5	1		1				1		1		1	
6		1		1				1		1	1	1
7	1		1		1				1		1	
8		1		1		1				1		1
9	1		1		1		1				1	
10		1		1		1		1				1
11	1		1		1	1	1		1			
12		1		1		1		1		1		



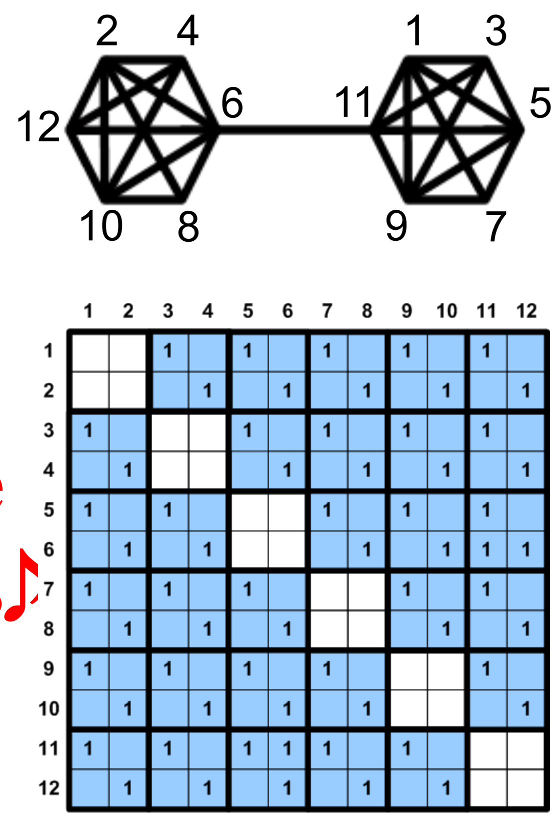
	1	2	3	4	5	6	7	8	9	10	11	12
1		1	1	1	1	1						
2	1		1	1	1	1						
3	1	1		1	1	1						
4	1	1	1		1	1						
5	1	1	1	1		1						
6	1	1	1	1	1		1					
7						1		1	1	1	1	1
8							1		1	1	1	1
9								1		1	1	1
10								1	1		1	1
11								1	1	1		1
12								1	1	1	1	

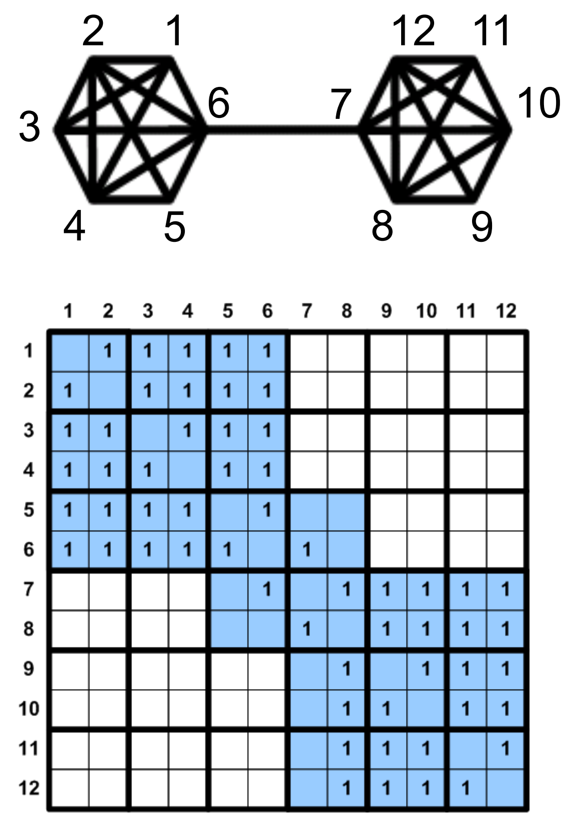
Carnegie Mellon

Good ordering = Good compression♪

- Same graphs with different orderings♪

Many sparse blocks♪



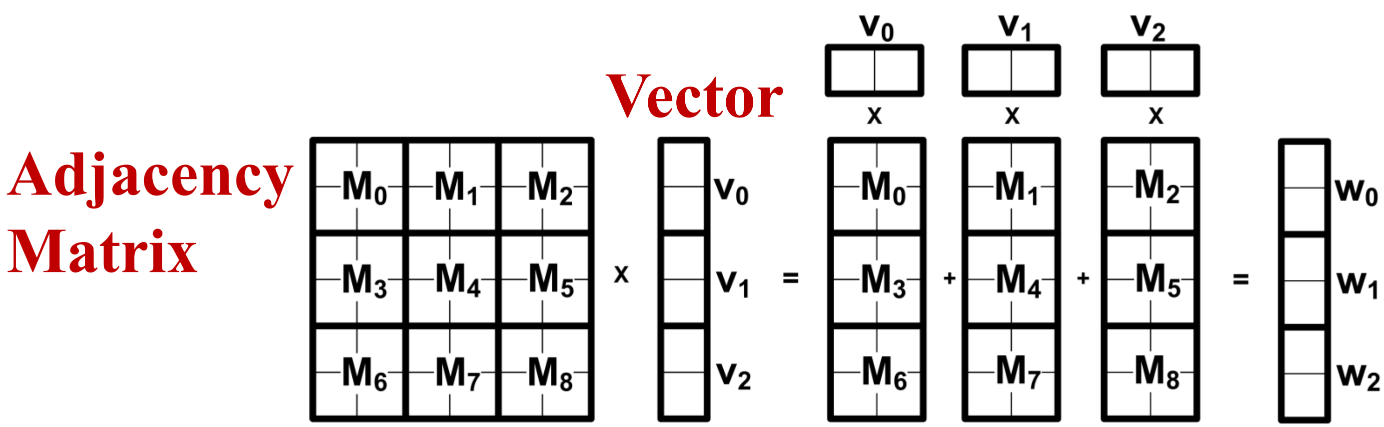


Few dense blocks♪

Carnegie Mellon

Application

- Block-based matrix-vector multiplication



Few, dense blocks

=> Better compression, faster running time

Problem Definition

- Given a graph, how can we lay-out its edges so that nonzero elements are well-clustered?
- Better clustering = better compression

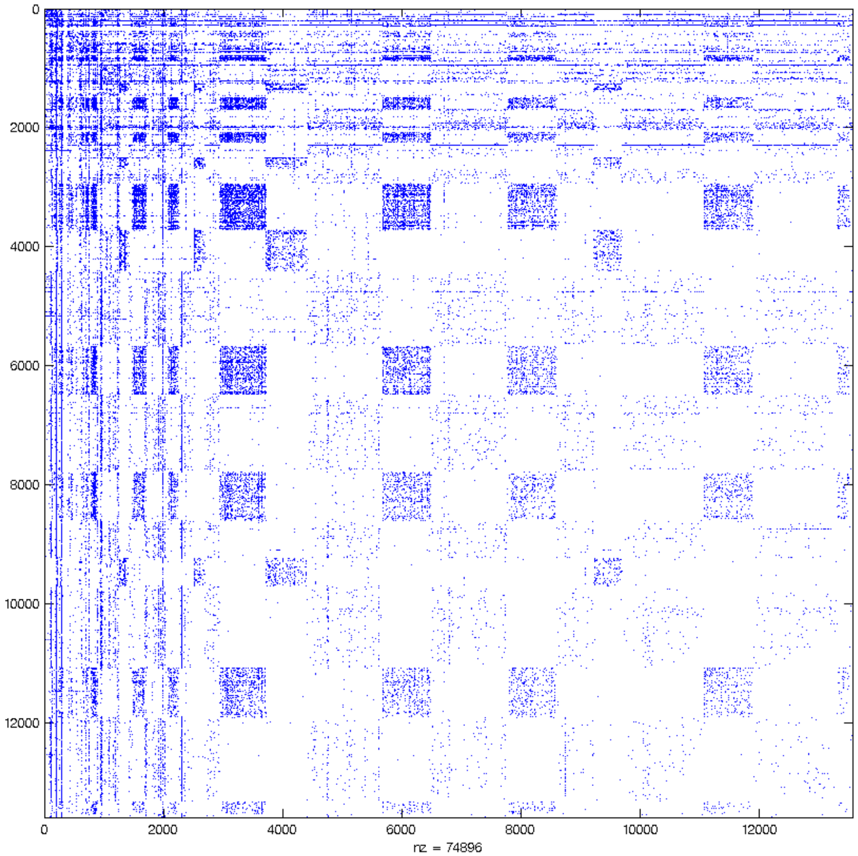
Many sparse blocks

	1	2	3	4	5	6	7	8	9	10	11	12
1			1		1		1		1		1	
2				1		1		1		1		1
3	1				1		1		1		1	
4		1				1		1		1		1
5	1		1				1		1		1	
6		1		1				1		1	1	1
7	1		1		1				1		1	
8		1		1		1				1		1
9	1		1		1		1				1	
10		1		1		1		1				1
11	1		1		1	1	1		1			
12		1		1		1		1		1		

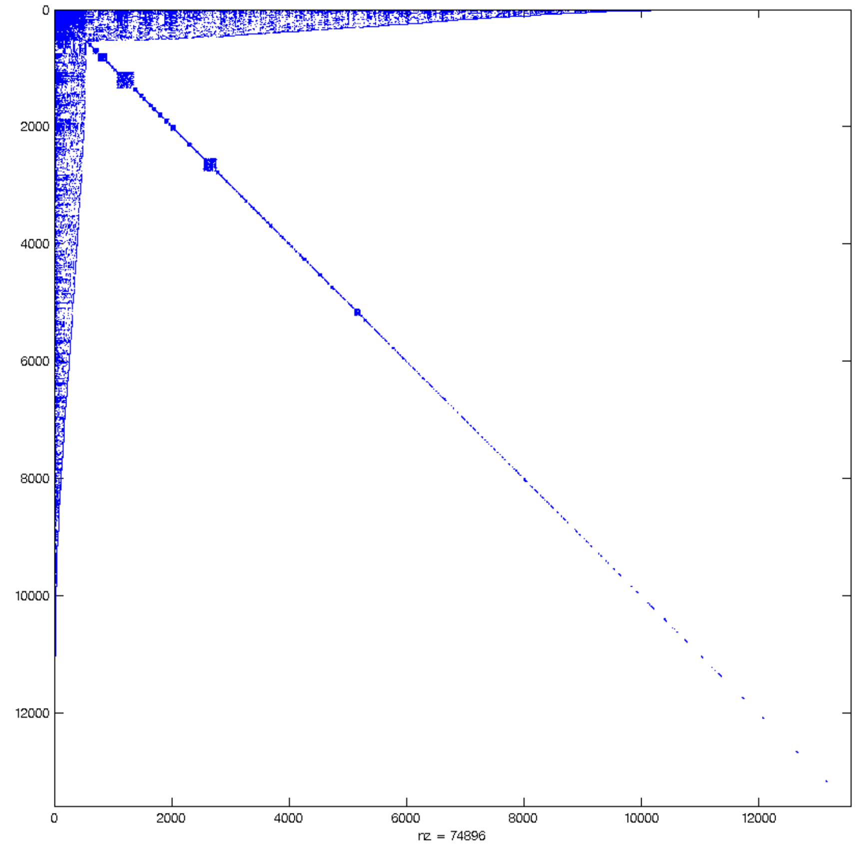
	1	2	3	4	5	6	7	8	9	10	11	12
1		1	1	1	1	1						
2	1		1	1	1	1						
3	1	1		1	1	1						
4	1	1	1		1	1						
5	1	1	1	1		1						
6	1	1	1	1	1		1					
7						1		1	1	1	1	1
8							1		1	1	1	1
9								1		1	1	1
10								1	1		1	1
11								1	1	1		1
12								1	1	1	1	

Few dense blocks

Main Result



Original



SlashBurn

Outline

- …
- Problem#3: Scalability - PEGASUS
  - Structure Analysis
  - Eigensolver
  - Graph Layout and Compression
    - Proposed Method
    - Results
- Conclusions

Carnegie Mellon

Survey

- Given a graph, how can we lay-out its edges so th at nonzero elements are well-clustered?

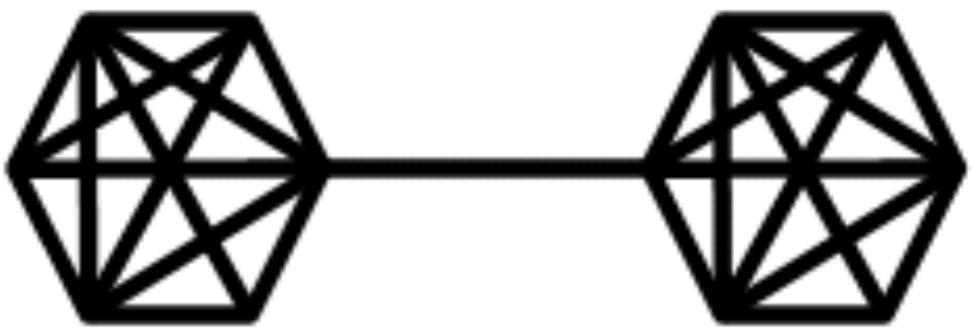
1) Graph based clustering
   - Normalized cut, spectral clustering

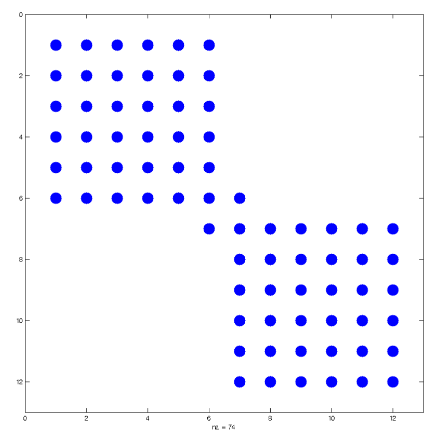
2) Heuristics
   - Lexicographic ordering for Web
   - Shingle ordering

Carnegie Mellon

1) Graph Based Clustering

- Goal: find homogeneous sets of nodes from graphs
  - E.g.) Spectral clustering and normalized cut
  - Many intra-edges, few inter-edges



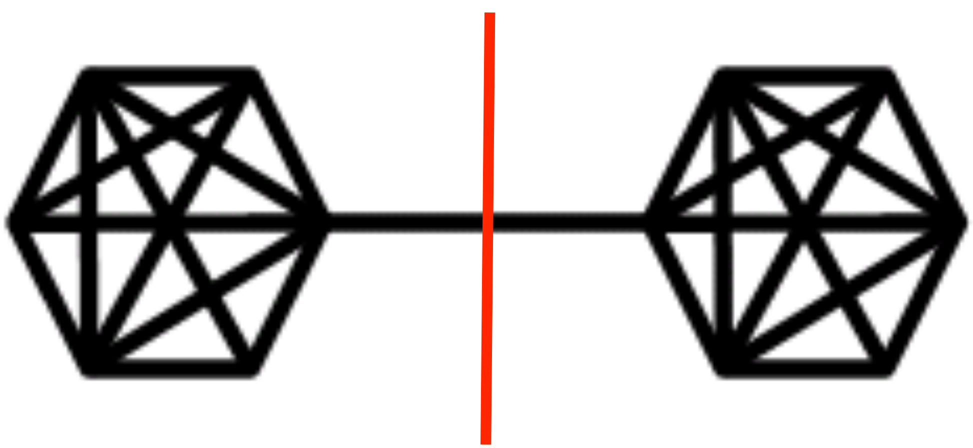


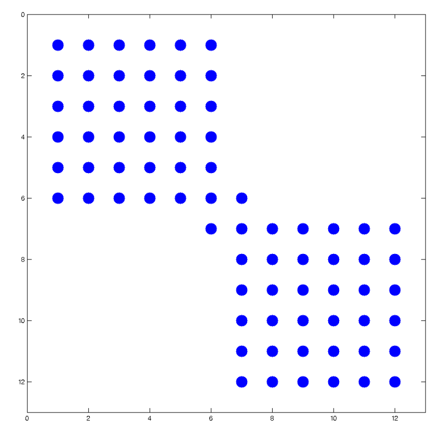
Caveman Communities

Carnegie Mellon

1) Graph Based Clustering

- Goal: find homogeneous sets of nodes from graphs
  - E.g.) Spectral clustering and normalized cut
  - Many intra-edges, few inter-edges

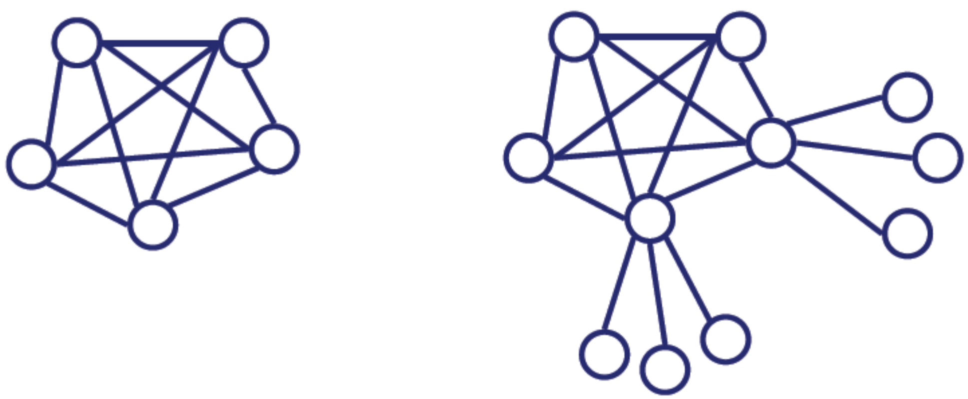




Caveman Communities

1) Graph Based Clustering

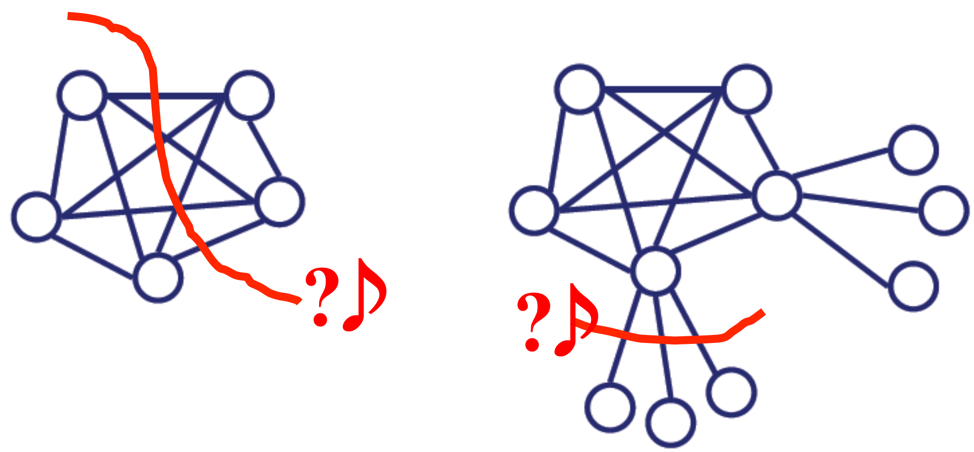
- But, real graphs: no good cuts
  - [Tauro+ 01], [Siganos+ 06], [Chakrabarti +04], [Leskovec+ 08]



Carnegie Mellon

1) Graph Based Clustering

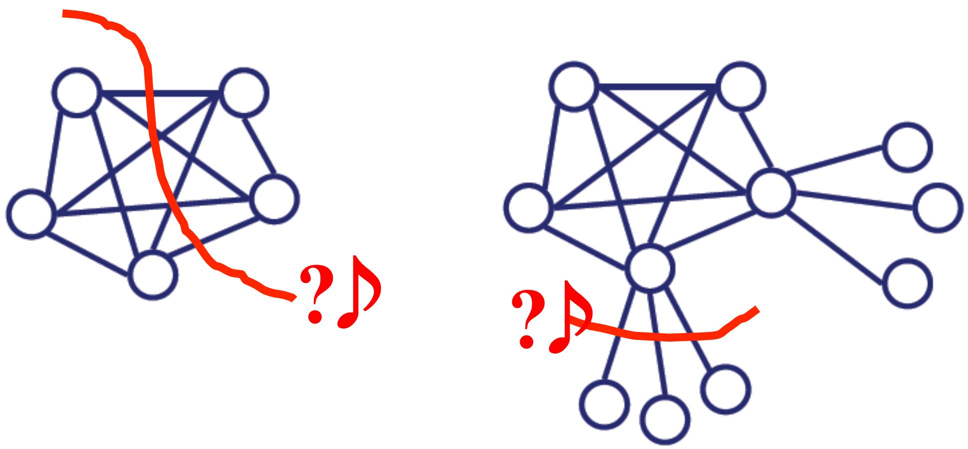
- But, real graphs: no good cuts
  - [Tauro+ 01], [Siganos+ 06], [Chakrabarti +04], [Leskovec+ 08]



Carnegie Mellon

1) Graph Based Clustering

- But, real graphs: no good cuts
  - [Tauro+ 01], [Siganos+ 06], [Chakrabarti +04], [Leskovec+ 08]



  - What should we do?

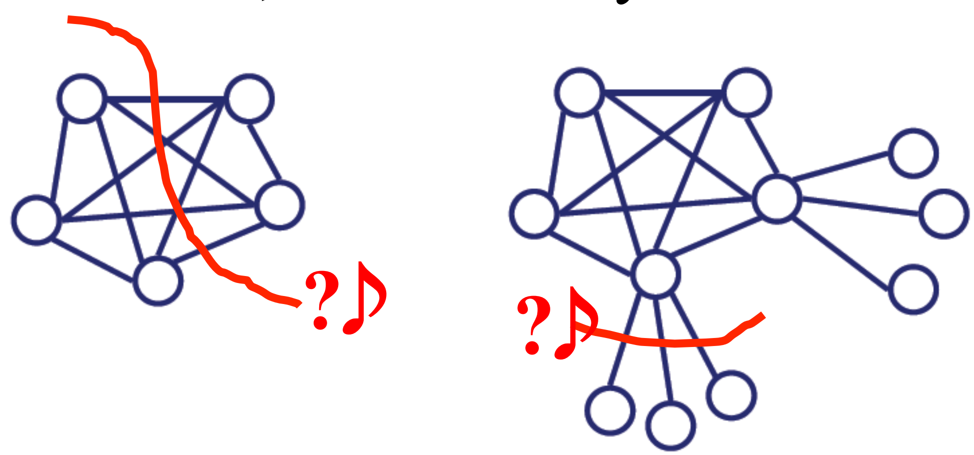
2) Older Heuristics

- Web graph: lexicographic ordering [Boldi+, 04]
  - Locality : many intra edges between neighbors
  - Similarity : out links of neighbors are similar

- Social network: shingle ordering [Chierichetti+ 09]
  - Group nodes with similar our-neighbors

Summary : Previous Works

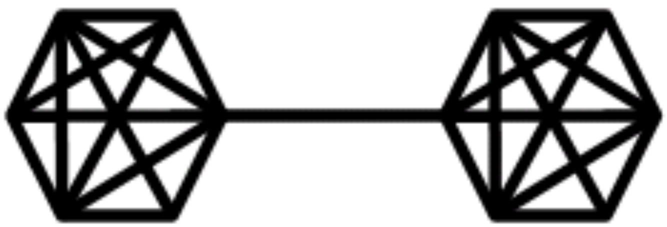
- Tries to find homogenous regions for graph compression
  - Fails to find them, because they often don't exist

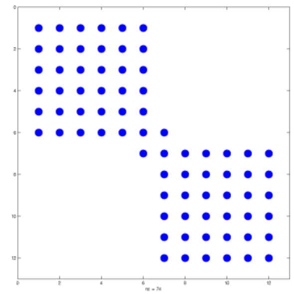


Carnegie Mellon

Our Observation

- Caveman assumption

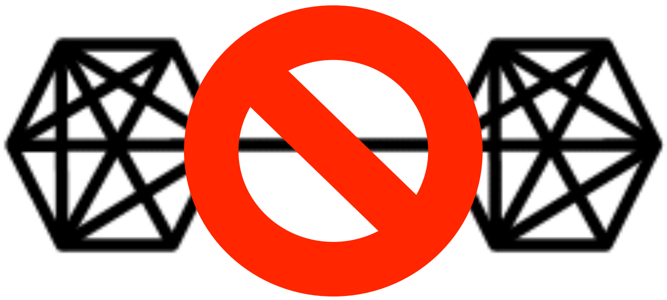


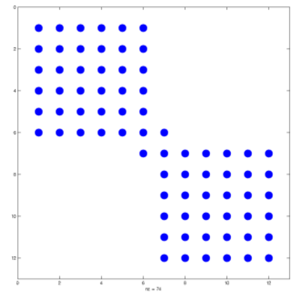


Carnegie Mellon

Our Observations

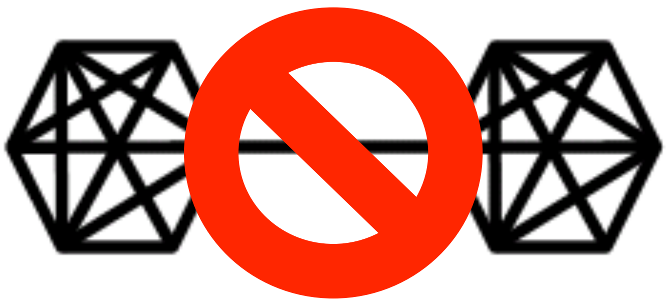
- Caveman assumption: wrong!

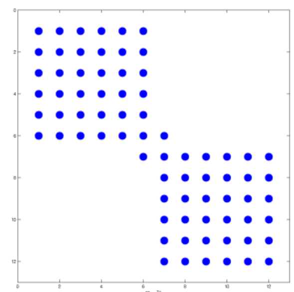




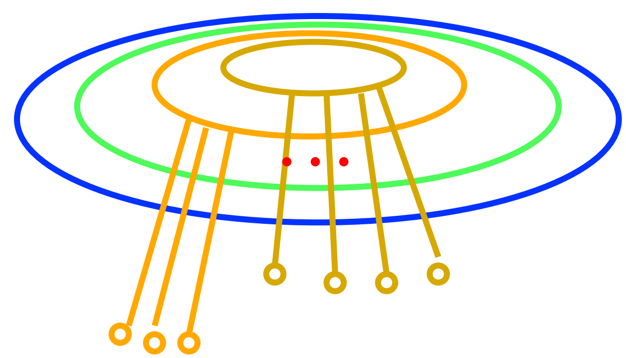
Our Solution

- Caveman assumption: wrong!





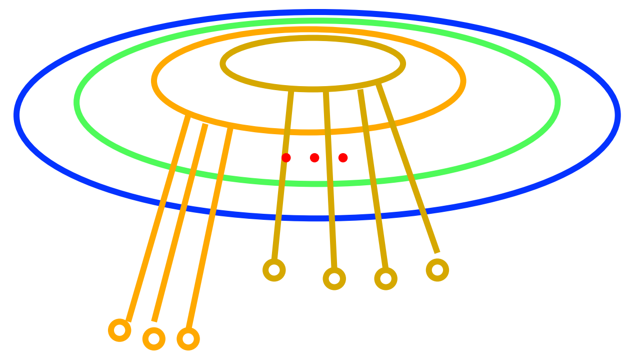
- Instead, we envision graphs as **nodes** connected by **connectors** connected by **super connectors**…



Carnegie Mellon

Our Solution♪

- Instead, we envision graphs as **nodes** connected by **connectors** connected by **super connectors**…

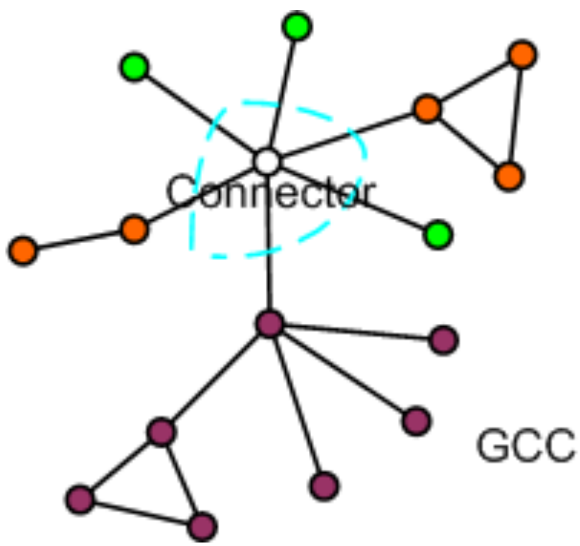


- Use "Graph Shattering" to `**peel**' the graphs from **super connectors**, and then **connectors**, …♪

Graph Shattering

- *k*-shattering of a graph G
  - Removes top k connectors and their incidents edges fro m G

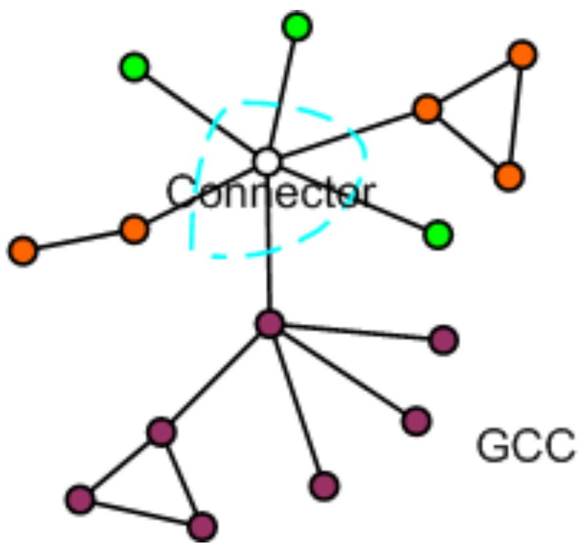
Graph Shattering



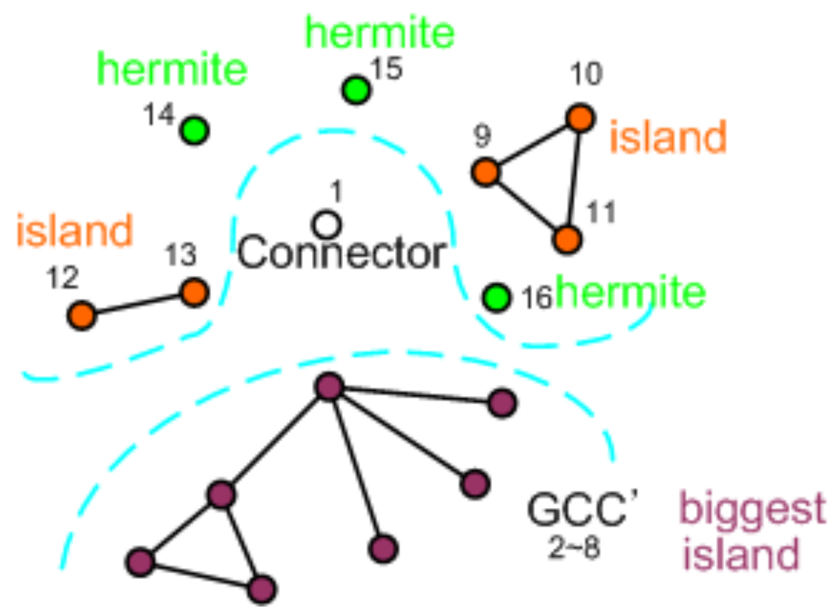
Before shattering

Carnegie Mellon

Graph Shattering

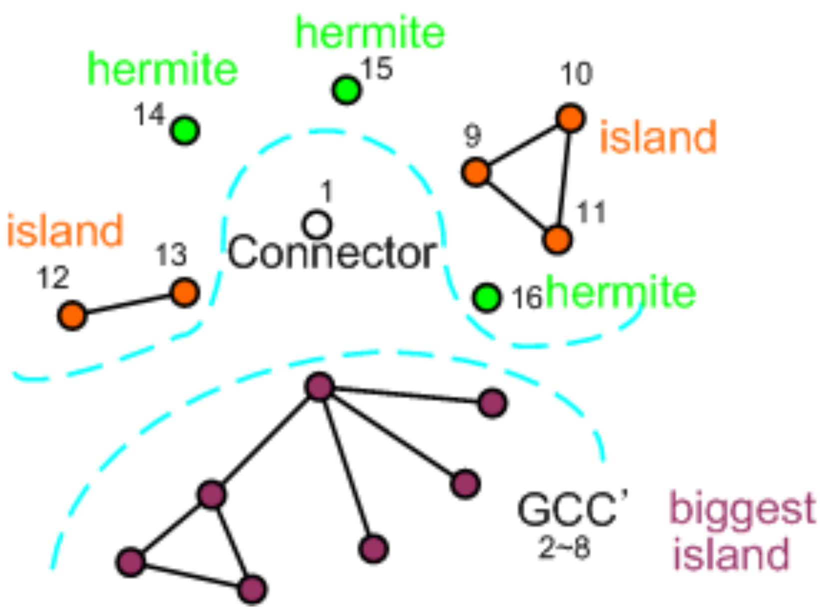


Before shattering

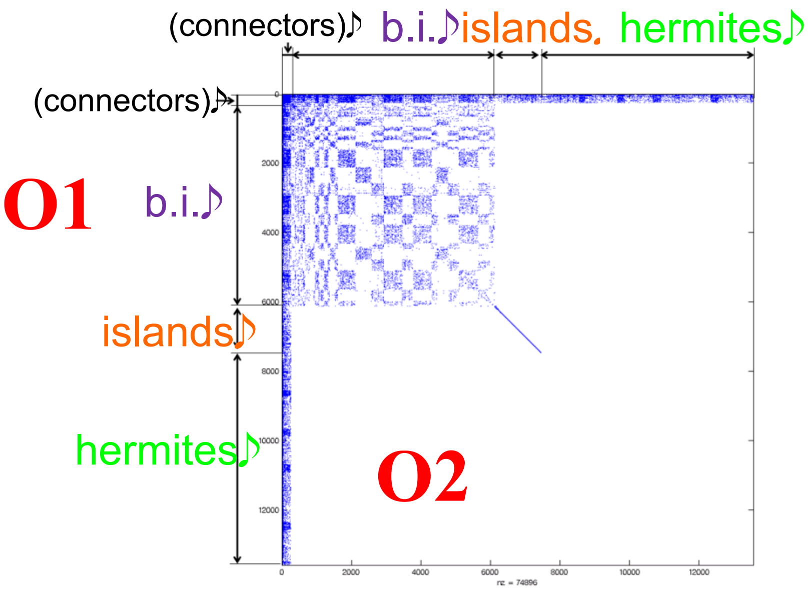


After shattering

Graph Shattering



After shattering



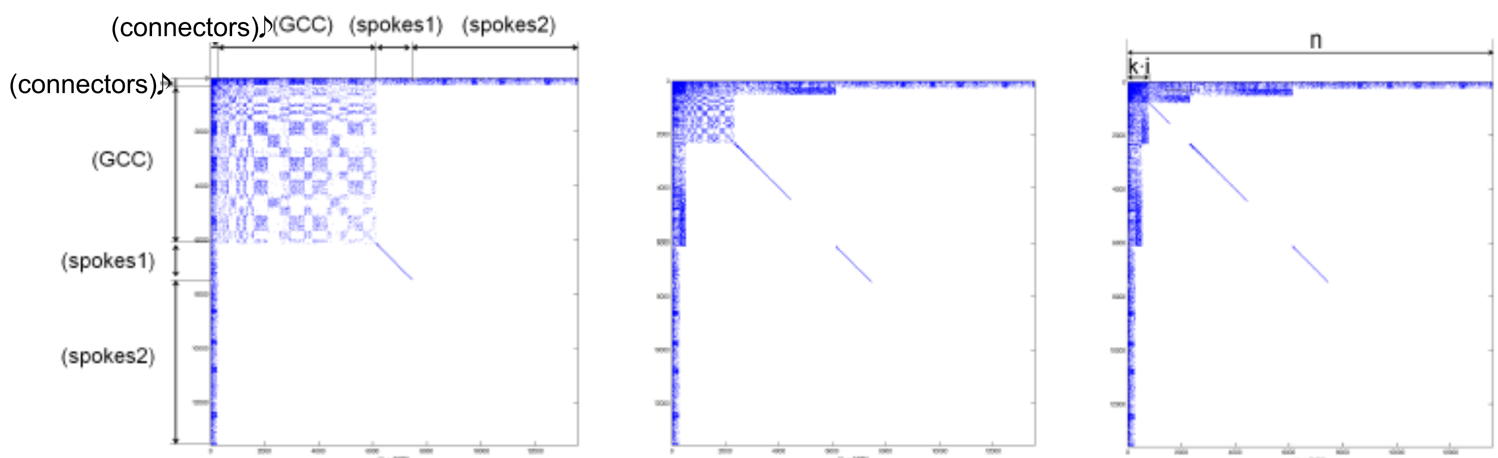
- Observations in real graphs

  O1. Portion of GCC is much smaller after shattering

  O2. A lot of disconnected components

Slash-Burn method (intuition)

- ‘burn’ the top k connectors, and ‘slash’ the edges
- Move k connectors to the front of the row/column,
  sort connected components by decr. size
- Recurse on the remaining GCC



(a) AS-Oregon after 1 iteration (b) .. after 1 more iteration (c) .. after 1 more iteration

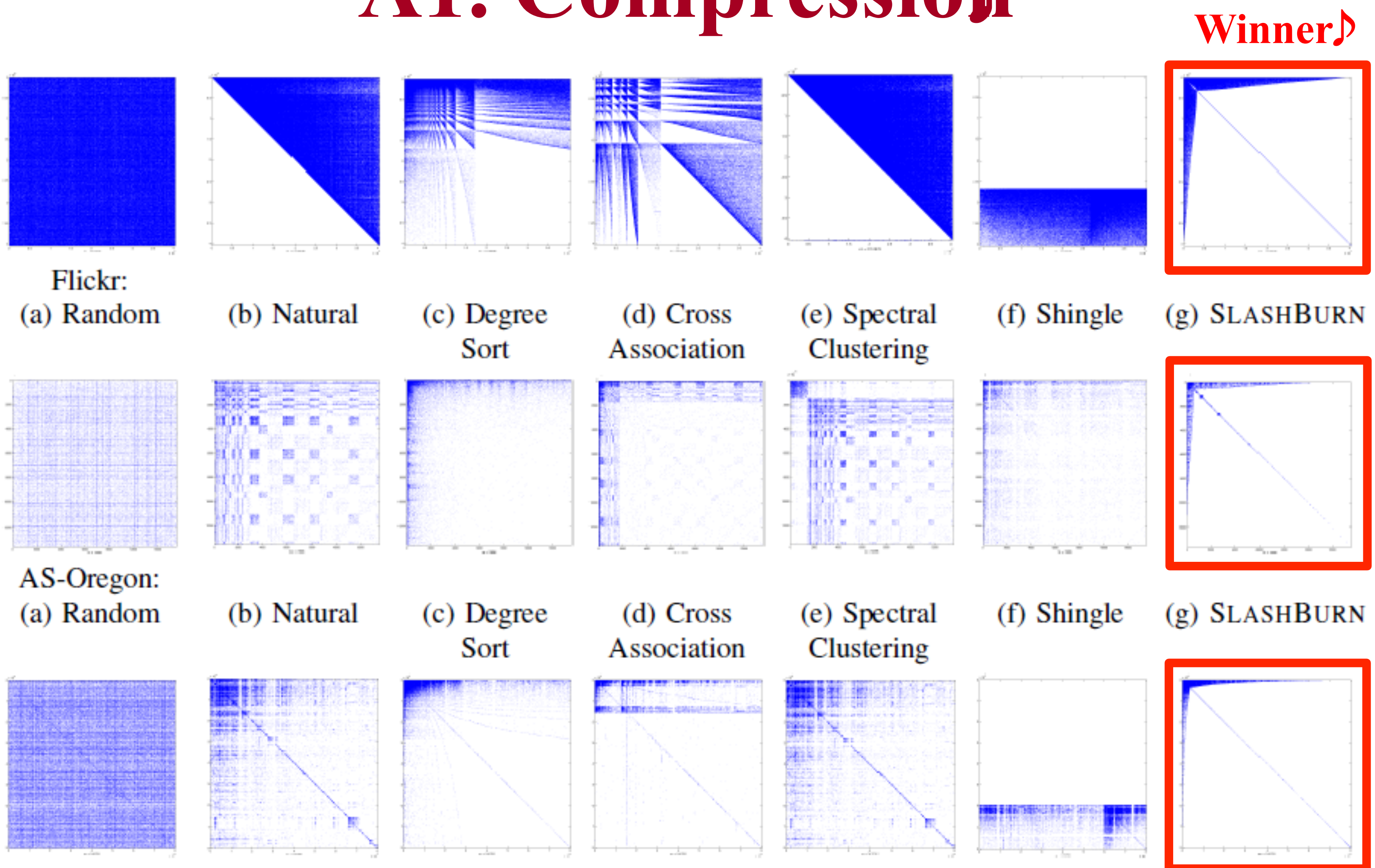
Outline

- …
- Problem#3: Scalability - PEGASUS
  - Structure Analysis
  - Eigensolver
  - Graph Layout and Compression
    - Proposed Method
    - Results
- Conclusions

Goal of Experiments

- [Q1] Compression savings?
- [Q2] Running time savings?

A1. Compression



Flickr:
(a) Random (b) Natural (c) Degree Sort (d) Cross Association (e) Spectral Clustering (f) Shingle (g) SLASHBURN

AS-Oregon:
(a) Random (b) Natural (c) Degree Sort (d) Cross Association (e) Spectral Clustering (f) Shingle (g) SLASHBURN

Enron:
(a) Random (b) Natural (c) Degree Sort (d) Cross Association (e) Spectral Clustering (f) Shingle (g) SLASHBURN

A1. Compression

Details

- Cost functions used
  1) Number of non-empty blocks
  2) Information theoretic cost : minimum bits to encode no nzero elements inside blocks

model complexity

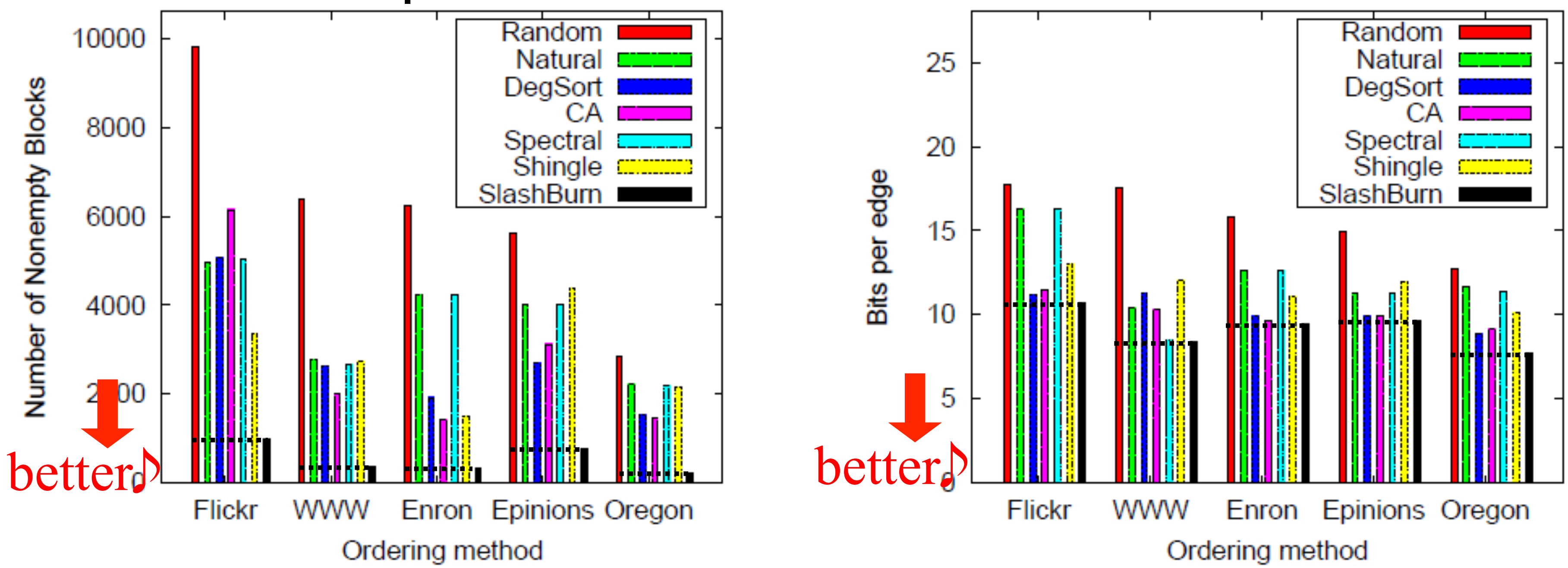
$$\boxed{|T| \cdot 2log\frac{n}{b}} + \boxed{\sum_{\tau \in T} b^2 \cdot H(\frac{z(\tau)}{b^2})}$$

costs given the model

|T|: # of nonempty blocks
n: # of nodes
b: block width
H(): Shannon entropy func

A1. Compression

- Cost comparison

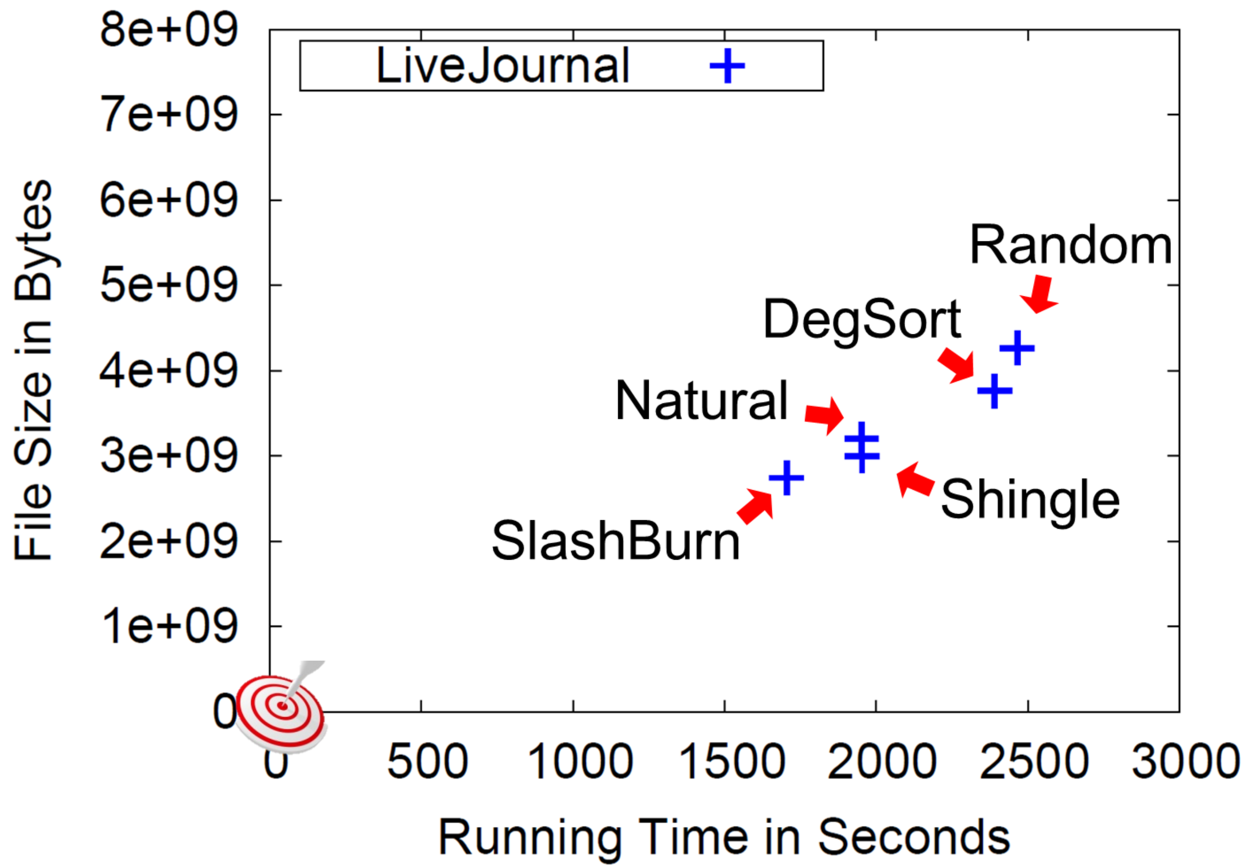


(a) $cost_{nz}(A, b)$: number of nonempty blocks (b) $cost_{it}(A, b)$: information theoretic cost

- SlashBurn outperforms all competitors for all dataset!
(smallest number of nonempty blocks, as well as bits per edge)

Carnegie Mellon

A2. Running Time



- SlashBurn outperforms all competitors !
(running time as well as file size)

Outline

- Introduction – Motivation
- Problem#1: Patterns in graphs
- Problem#2: Tools
- Problem#3: Scalability - PEGASUS
- Conclusions

Conclusions

- PEGASUS: Peta-Scale Graph Mining System
  - Patterns and anomalies in large graphs
    - PageRank, Connected Components, Radius, Eigensolver
  - Outreach
    - Microsoft : part of Hadoop distribution for Windows Azure
    - One of the core systems for several DARPA projects (ADA MS, INARC, DTRA)

PROJECT PEGASUS

Open Source Software World Challenge

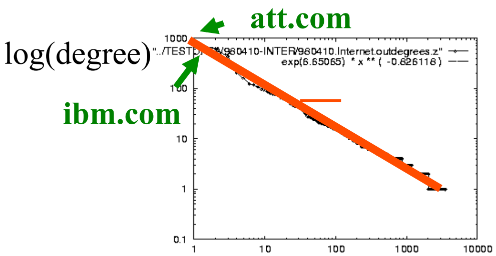
www.cs.cmu.edu/~pegasus

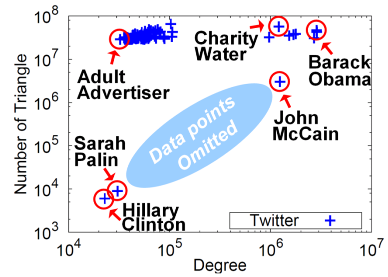
Conclusions

- High impact applications require large graph mini ng

Cyber Security	Fraud Detection	Social Network	Search Engine	Health Care

Carnegie Mellon





PROJECT PEGASUS

www.cs.cmu.edu/~pegasus

Thank you !

Complementary tutorial: *Mining Billion-Scale Graphs: Systems and Implementations*: Haixun Wang et al